Klausur zur Vorlesung
„Konzepte und Methoden der Systemsoftware“
Uni Paderborn SS 00
Prof. Dr. H.-U. Heiß
2. Oktober 2000

Name, Vorname	
Matrikelnummer	
Fachbereich	
Studiengang	
Prüfungsordnung	
Prüfung oder Leistungsnachweis?	

- *Schreiben Sie zunächst sofort Ihren Namen und Ihre Matrikelnummer auf jedes Blatt der Klausur!*
- *Blätter ohne Namen werden nicht gewertet!*
- *Lassen Sie die Klausur zusammengeheftet!*
- *Die Klausur dauert 120 Minuten und umfasst 6 Aufgaben auf 16 Seiten.*
- *Die Klausur ist bestanden, wenn mindestens 50% der Punkte erreicht wurden.*
- *Es sind keine Hilfsmittel zugelassen (insbesondere keine Taschenrechner und Handys)!*
- *Abschreiben und abschreiben lassen führt zum Nichtbestehen der Klausur!*
- *Benutzen Sie kein eigenes Konzeptpapier bzw. Schmierpapier. Sie bekommen bei Bedarf Papier von der Klausuraufsicht!*
- *Wenn Sie Konzeptpapier abgeben wollen, dann nur mit Namen und Matrikelnummer!*
- *Kennzeichnen Sie Ihre Lösung eindeutig. Es wird keine Lösung gewertet, wenn Sie zu einer Aufgabe mehr als eine Lösung abgeben.*

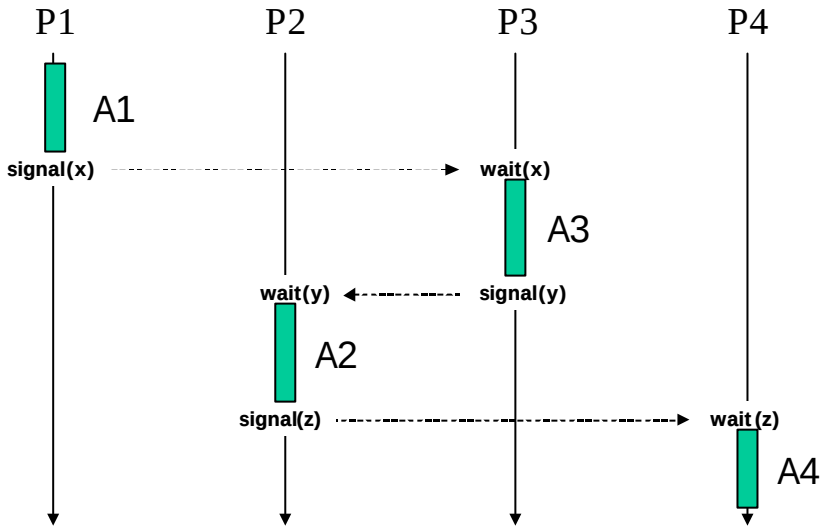
Aufgabe	1	2	3	4	5	6	Σ
Punkte	24	20	22	10	11	33	120
Erreicht							

Note	Bestanden

Aufgabe 1: Koordination von Prozessen (4+4+16 =24 Punkte)

a) Signalisierung (4 Punkte)

Verwenden Sie die Signalisierungsoperationen zur Synchronisation nebenläufiger Prozesse, um im angegebenen Schaubild ein Szenario zu beschreiben, bei dem die Abschnitte Ai der Prozesse Pi , i = {1,…,4}, in der folgenden Reihenfolge ausgeführt werden: A1, A3, A2, A4. Beachten Sie die Synchronisationsvariable(n)! Markieren Sie hierbei die jeweils zu einer Zeit aktiven Prozesse mit einem Balken über der Zeitachse (siehe rechts).



b) Unteilbare Aktionen (2+2 = 4 Punkte)

i. Wie realisiert man eine unteilbare Aktion mit Unterbrechungssperren?
Geben Sie ein Beispiel an! (2 Punkte)

Besitzt ein Prozessor einen Unterbrechungsmechanismus, so kann für die Dauer der kritischen Operation die Unterbrechungssperre gesetzt werden.

Die zu sichernde Aktion (Operation) wird durch *disable interrupt* und *enable interrupt* geklammert

– disable interrupt – kritische Operation – enable interrupt →

ii. Was sind unteilbare Maschinenoperationen?
Warum werden sie benötigt? (2 Punkte)

Eine unteilbare Maschinenoperation ist ein Maschinenbefehl, der in einer unteilbaren Operation den Wert einer Bedingungsvariable abfragt und gleichzeitig setzt (z.B. test-and-set).

In Mehrprozessorsystemen ist die Unterbrechungssperre nicht ausreichend, es kann trotz eines Unterbrechungsverbots zu einer verzahnten Ausführung von kritischen Operationen kommen.

Kritischer Abschnitt (6+2+8 = 16 Punkte)

Gegeben sei der folgende Modul zur Realisierung eines gegenseitigen Ausschlusses zwischen zwei Prozessen in der Notation der Vorlesung. Die Operation *sleep* versetzt den jeweiligen Prozess für eine bestimmte Zeit in einen Wartezustand.

```
module lockunlock;

    export LOCK1, UNLOCK1, LOCK2, UNLOCK2;

    var   crit1, crit2 : lockVariable = reset;        // lockVariable = {set, reset}


    procedure LOCK1;
      begin
          crit1 := set;
          while crit2 = set do
              crit1:= reset;
              sleep;
              crit1:= set;
          end; //while
      end;


    procedure UNLOCK1;
      begin
          crit1 := reset;
      end;
```

```
    procedure LOCK2;
      begin
          crit2 := set;
          while crit1 = set do
              crit2:= reset;
              sleep;
              crit2:= set;
          end; //while
      end;


    procedure UNLOCK2;
      begin
          crit2 := reset;
      end;

end lockunlock.
```

i. Welche der folgenden Probleme können bei der angegebenen Implementierung auftreten bzw. nicht auftreten. (Beachten Sie, dass falsche Antworten zu einem Punktabzug führen; die Summe der Punkte dieser Teilaufgabe ist ≥ 0.) (6 Punkte)

Problem kann auftreten:	ja	nein
Unterbrechung zwischen Überprüfen und Verändern der Lock-Variable	☐	☒
Verklemmung	☐	☒
Aushungern	☒	☐

ii. Beschreiben Sie die grundsätzliche Vorgehensweise, um alle der unter i. aufgeführten und durch die Implementierung nicht gelösten Probleme zu vermeiden! (2 Punkte)

Es gibt zwei alternative Möglichkeiten, um das Aushungern zu vermeiden:

1. **Prioritätszähler** einführen, die die crit-Variablen ersetzen
2. Variable einführen, die den alternierenden Eintritt in den kritischen Abschnitt sicher stellt. Es ist allerdings nur dann sinnvoll, Prozesse abwechselnd den Vortritt zu lassen, wenn wirklich beide den Eintrittswunsch in den kritischen Abschnitt haben, d.h. ein Konflikt auftritt. Deshalb verwenden wir eine **Favorisierung** des Prozesses, der nicht als letzter im kritischen Abschnitt war.

iii. Ergänzen Sie den oben angegebenen Modul um die in ii. genannten Konzepte, so dass alle der unter i. angegebenen Probleme verhindert werden (Hinweis: Nehmen Sie Veränderungen, Einfügungen, Streichungen etc. an dem auf der vorherigen Seite gegebenen Programmtext vor.) (8 Punkte)

1. Lösungsmöglichkeit mit *Prioritätenzähler*

```
module lockunlock;

  export LOCK1, UNLOCK1, LOCK2, UNLOCK2;

  var   prio1, prio2 : integer = 0;

  procedure LOCK1;
    begin
      prio1 := prio2 + 1;
      while (prio2 <> 0 and
             prio2 < prio1) do
        sleep;
      end;
    end;

  procedure UNLOCK1;
    begin
      prio1 := 0;
    end;

  procedure LOCK2;
    begin
      prio2 := prio1 + 1;
      while (prio1 <> 0 and
             prio1 < prio2) do
        sleep;
      end;
  end;

  procedure UNLOCK2;
    begin
      prio2 := 0;
    end;

end lockunlock.
```

2. Lösungsmöglichkeit mit *Favorisierung*

```
module lockunlock;

    export LOCK1, UNLOCK1, LOCK2, UNLOCK2;

    var   crit1, crit2 : lockVariable = reset;          // lockVariable = {set, reset}

          favorit : {1,2} = 1;

    procedure LOCK1;
       begin
          crit1 := set;
          while crit2 = set do
               if favorit <> 1 then
                 begin
                    crit1:= reset;
                    while favorit <>1 do
                       sleep;
                    end;
                    crit1:= set;
                 end;
          end;
       end;

    procedure UNLOCK1;
       begin
          favorit := 2;
          crit1 := reset;
       end;

    procedure LOCK2;
       begin
          crit2 := set;
          while crit1 = set do
               if favorit <> 2 then
                 begin
                    crit2:= reset;
                    while favorit <>2 do
                       sleep;
                    end;
                    crit2:= set;
                 end;
          end;
       end;

    procedure UNLOCK2;
       begin
          favorit := 1;
          crit2 := reset;
       end;

end lockunlock.
```

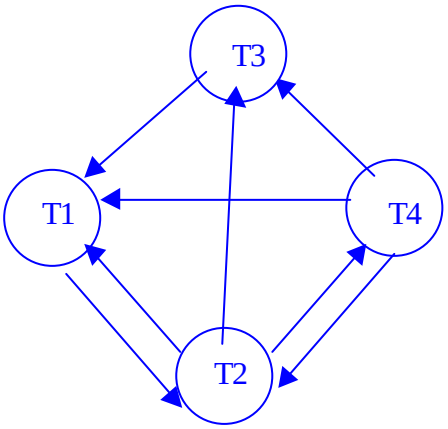
Aufgabe 2: Transaktionen (8+12 = 20 Punkte)

Gegeben sei die folgende Ausführungsfolge von vier nebenläufigen Transaktionen

Op	T1	T2	T3	T4
1.			write(c)	
2.				write(b)
3.	read(b)			
4.	read(c)			
5.		read(a)		
6.		write(a)		
7.	write(c)			
8.	write(a)			
9.		read(d)		
10.		write(b)		
11.				write(d)
12.				commit
13.			write(d)	
14.			commit	
15.	commit			
16.		commit		

a) Serialisierbarkeitsgraph (8 Punkte)

Zeichnen Sie den Serialisierbarkeitsgraphen!



b) Eigenschaften (12 Punkte)

Geben Sie an, welche Eigenschaft der durch diese Ausführungsreihenfolge definierte Plan besitzt und begründen Sie kurz Ihre Antworten.

i) serialisierbar ❑ ja ☒ nein

Begründung:

Der Serialisierbarkeitsgraph (siehe a)) enthält einen Zyklus

ii) rücksetzbar ☒ ja ❑ nein

Begründung:

T1 liest von und bestätigt nach T3

T1 liest von und bestätigt nach T4

T2 liest von und bestätigt nach T1

iii) kaskadierenden Abbruch vermeidend ❑ ja ☒ nein

Begründung:

Es werden unbestätigte Daten gelesen (Op 3 und 4; Dirty Read)

iv) unbestätigte Daten überschreibend ☒ ja ❑ nein

Begründung:

Es werden unbestätigte Daten überschrieben (Op 8, 10 und 13; Dirty Write)

Aufgabe 3: Betriebsmittelverwaltung (8+6+8 = 22 Punkte)

a) Sicherheit (4+4 = 8 Punkte)

Gegeben sei ein System mit $m = 3$ Prozessen ($i = 1,..,3$) und $n = 2$ Betriebsmitteltypen ($j = 1,2$). Ein Teil der vorhandenen Betriebsmittel sei belegt, ein Teil sei noch frei. Außerdem seien die Restanforderungen der Prozesse bekannt. Die Situation sei durch die folgenden Größen beschrieben:

Belegungen: Restanforderungen: frei:

$$B=\begin{pmatrix}2 & 3\\ 1 & 0\\ 8 & 4\end{pmatrix} \qquad R=\begin{pmatrix}1 & 1\\ 3 & 0\\ 4 & 6\end{pmatrix} \qquad f=\begin{pmatrix}1 & 2\end{pmatrix}$$

i. Wenden Sie den Banker-Algorithmus an, um zu überprüfen, ob das System sicher ist! (4 Punkte)

1) r1 = (1 1) < f = (1 2) => p1 kann beendet werden => f‘ =(3 5)
2) r2 = (3 0) < f‘ => p2 kann beendet werden => f“=(4 5)
3) r3 = (4 6) > f“ => System unsicher

ii. Wenn das System *sicher* ist, überlegen Sie, welche und wie viele Betriebsmittel man mindestens entfernen muss, damit das System unsicher wird.

Wenn das System *unsicher* ist, geben Sie an, wie viele freie Betriebsmittel ab dem oben beschriebenen Zustand gebraucht werden, damit das System sicher wird.

(4 Punkte)

System ist unsicher:

f = (f1 f2) = (1 2)

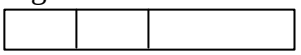
1) f1 = 1 ist nötig, weil nicht zu groß, daher braucht hier nicht minimiert zu werden
2) f2 = 2 ist zu klein, es muss wegen (4 6) > (4 5) um eins erhöht werden

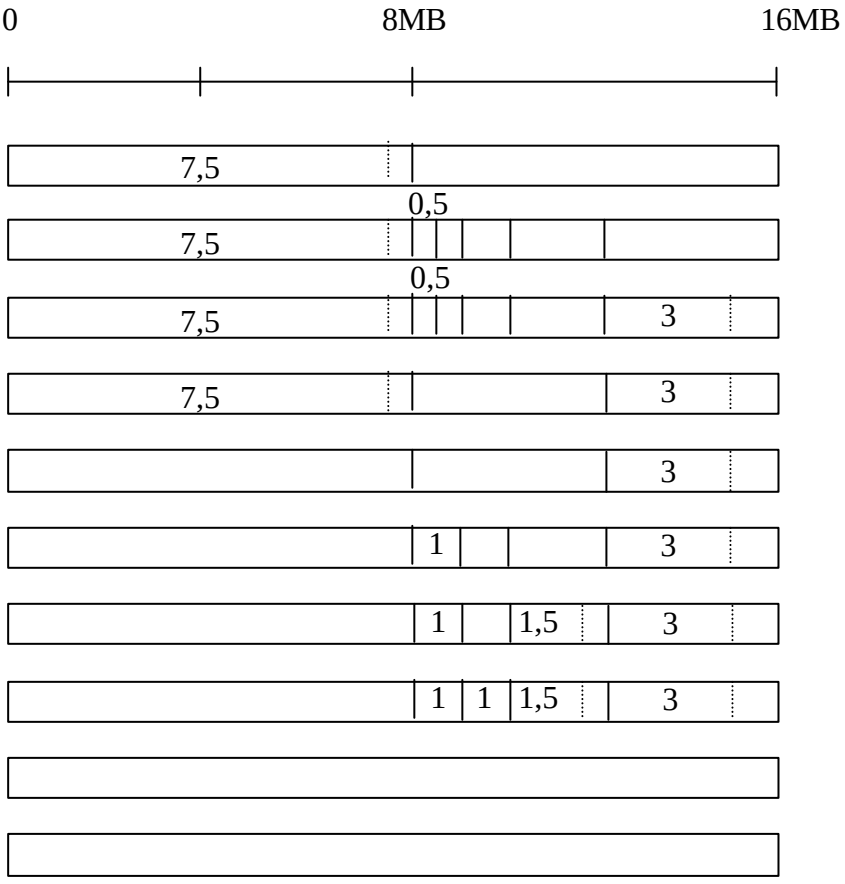
=> f neu = (1 3)

b) Buddy-Verfahren (4+2 = 6 Punkte)

Ein 16 MByte großer Speicher werde nach dem Buddy-Verfahren verwaltet. Auf dem zu Beginn leeren Speicher werden der Reihe nach die folgenden Operationen durchgeführt:

1. Anforderung 1: 7,5 MB
2. Anforderung 2: 0,5 MB
3. Anforderung 3: 3 MB
4. Freigabe Anforderung 2 (0,5 MB)
5. Freigabe Anforderung 1 (7,5 MB)
6. Anforderung 4: 1 MB
7. Anforderung 5: 1,5 MB
8. Anforderung 6: 1 MB

i. Können alle Anforderungen erfüllt werden? Geben Sie die Anordnung im Speicher an! Nutzen Sie dafür die unten angegebenen rechteckigen Felder. Deuten Sie Teilungen des Speichers durch senkrechte Striche an, z.B.:  (4 Punkte)



ii. Geben Sie die kleinste mögliche Anforderung ≤ 16 MB als Operation 9 an, die nicht erfüllt werden kann! (2 Punkte)

Z.B. 8 MB +1 Bit.

Z.B. 8 MB + 1 Byte. Dann muß bei der Lösung darauf hinweisen, was man unter der kleinsten Einheit versteht, z.B 1 Byte, sonst kann man nicht minimieren.

c) Interner Verschnitt (8 Punkte)

Nehmen Sie an, die Speicheranforderungen seien von der Art 2^k KByte, k sei gleichverteilt im Intervall [0, 10].

Die Speicherverwaltung stellt vorkonfektionierte Stücke der Größen 64 Kbyte, 256 KByte und 1 MByte zur Verfügung. Andere Größen gibt es nicht.

Welcher mittlere interne Verschnitt stellt sich ein?

Es reicht einmal alle möglichen Anforderungen und den dabei entstehenden Verschnitt zu betrachten, da die Anzahl der möglichen Anforderungen endlich ist und die möglichen Vergaben auch und diese sich nicht ändern.

2^0 bis 2^6 KB => Vergabe von 7* 64KB

2^7 und 2^8 KB => Vergabe von 2* 256KB

2^9 und 2^10 KB => Vergabe von 2* 1 MB

Verschnitt = nicht verwendet aber belegt / belegte Blöcke =

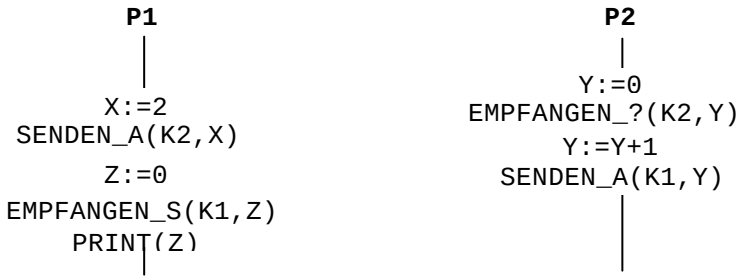
$\frac{63+62+60+56+48+32 + 128 + 512}{7*64 + 2*256 + 2* 1024}$

Aufgabe 4: Kommunikation nebenläufiger Prozesse (5+5 = 10 Punkte)

Gegeben seien die beiden unten skizzierten miteinander kommunizierenden Prozesse P1 und P2. Als Nachrichten werden ganzzahlige Werte (integer) ausgetauscht: Senden(K,N) legt den Wert der Variablen N im Kanal K ab, während Empfangen(K,N) den im Kanal K abgelegten Wert in die Variable N kopiert.

Wir nehmen nun an, dass die Empfangsoperation in P2 vor der entsprechenden Sendeoperation in P1 aufgerufen wird. Ansonsten sei eine beliebige Verzahnung der Prozesse möglich. Die Kanäle seien zu Beginn leer.

Das Ergebnis der Ausführung hängt nun ab von der Art der Empfangsoperation (synchron / asynchron) in P2, die noch offen gelassen wurde (`EMPFANGEN_?`).



Welcher Wert wird durch P1 ausgedruckt, wenn es sich um die folgende Empfangsart handelt (bitte ankreuzen):

a) Synchron (5 Punkte)

`EMPFANGEN_S(K2,Y)`

❑ 0 ❑ 1 ❑ 2 **X** 3 ❑ unbestimmt ❑ es wird nichts gedruckt, da P1 blockiert

b) Asynchron (5 Punkte)

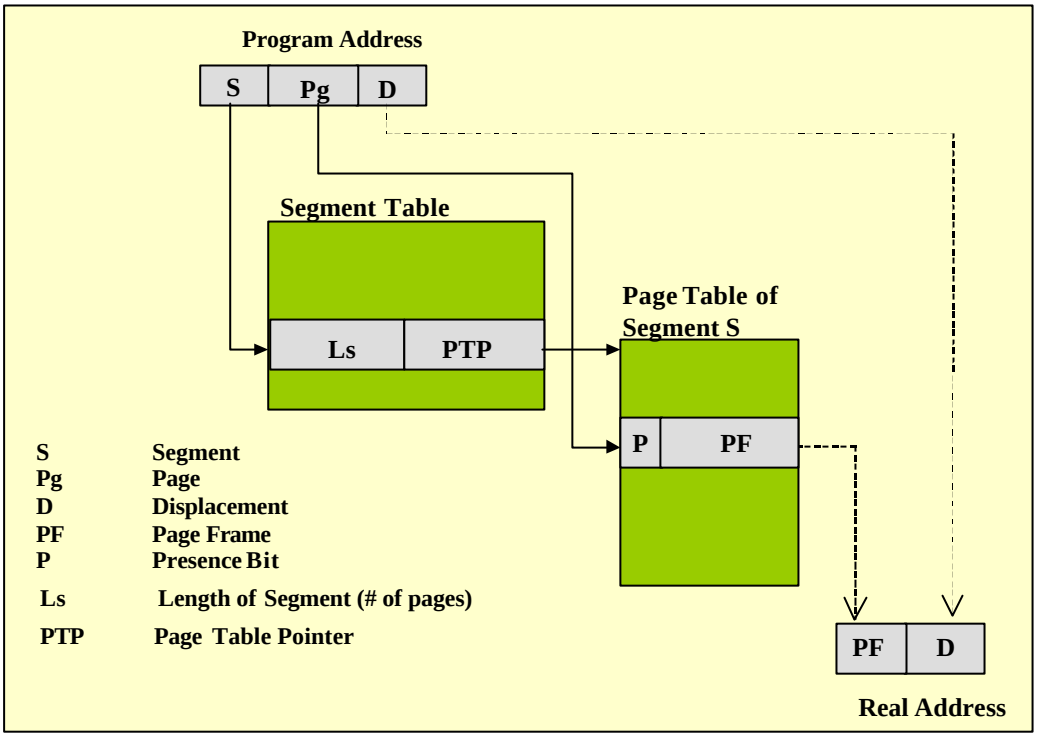
`EMPFANGEN_A(K2,Y)`

❑ 0 ❑ 1 ❑ 2 **X** 3 unbestimmt ❑ es wird nichts gedruckt, da P1 blockiert

Aufgabe 5: Speicherverwaltung (6+5 = 11 Punkte)

a) Adressumsetzung (6 Punkte)

Gegeben sei ein virtuelles Speichersystem mit dem nachfolgenden Adressumsetzungsmechanismus.



Berechnen Sie für jede Spalte der unten stehenden Tabelle die *reale Adresse* bzw. das Auftreten eines Segment- oder Seitenfehlers.

	A 1	A 2	A 3
Pg	10 101	100 111	1 111 001
D	010	111	101
Ls	11 000	100 000	100 000 000
P	1	1	0
PF	010	011	111
Reale Adresse/ Fehler	*010 010*	*Segmentfehler*	*Seitenfehler*

Hinweis: Ein Segmentfehler (segmentation fault) tritt ein, wenn ein Speicherzugriff außerhalb des referenzierten Segmentes versucht wird. Ein Seitenfehler ist ein Zugriff auf eine Seite, die zum aktuellen Zeitpunkt nicht im realen Speicher verfügbar ist.

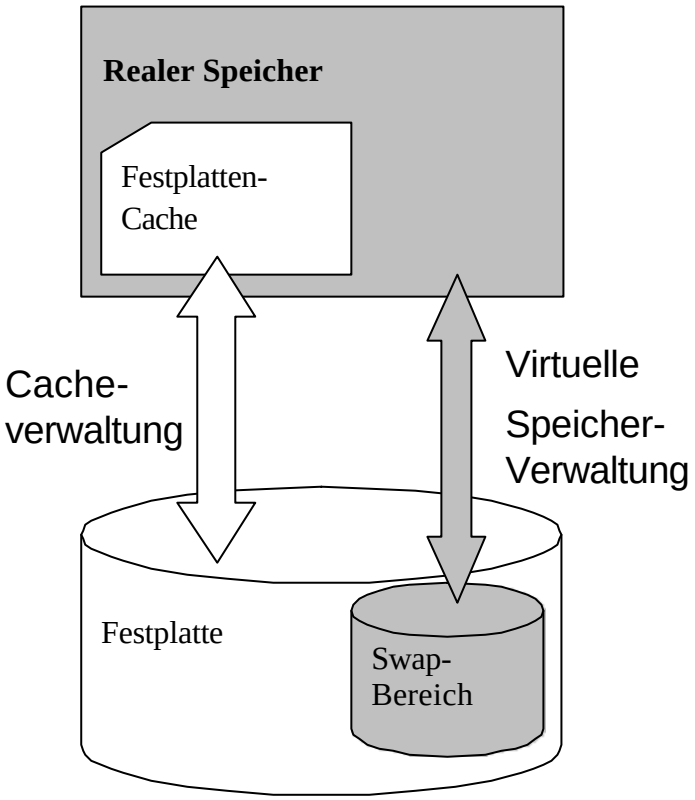
b) Zugriffszeit (5 Punkte)

Auf einem PC-System wird von Seiten des Benutzers ein zusätzlicher Festplattencache installiert, der wie andere Anwendungsprogramme der virtuellen Speicherverwaltung unterworfen ist. Wie lange dauert im schlimmsten Fall ein Cachezugriff, wenn die reale Zugriffszeit der Festplatte bei 10ms und die reale Speicherzugriffszeit bei 50ns liegt. (Anmerkung: Zur Vermeidung des hier auftretenden Problems ist es in realen Systemen möglich, Teile des Adressraums eines Anwendungsprogramms vor der Auslagerung zu schützen.)

Worst case: 20,0001 ms

1. *Cachetabelle ist ausgelagert, d.h.: 50 ns + 10 ms*

Die gesuchten Daten befinden sich nicht im Cache, d.h.: 50 ns + 10 ms



Aufgabe 6: Scheduling (15+18 = 33 Punkte)

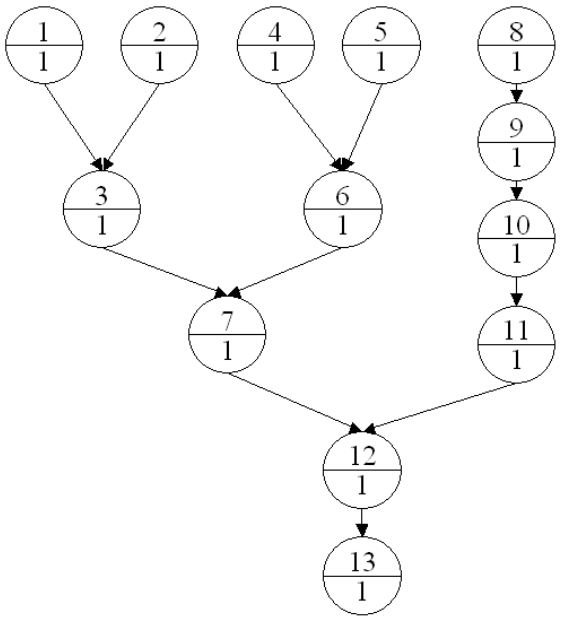
a) Instruction Scheduling (6+4+5 = 15 Punkte)

Gegeben sei folgender Programmcode:

1: R_1:=a;
2: R_2:=b;
3: R_3:=R_1*R_2;
4: R_4:=c;
5: R_5:=d;
6: R_6:=R_4*R_5;
7: R_7:=R_6+R_3;
8: R_8:=c;
9: R_9=R_8*R_8;
10: R_{10}=R_9*R_9;
11: R_{11}=R_{10}*R_{10};
12: R_{12}:=R_7+R_{11};
13: x=R_{12};

Jede Instruktion hat eine Bearbeitungszeit von 1.

i. Zeichnen Sie den zugehörigen Abhängigkeitsgraphen. (6 Punkte)



ii. Ermitteln Sie den Plan für ein System mit 3 Prozessoren mit dem List-Scheduling-Algorithmus aus der Vorlesung. Nehmen Sie dazu an, dass die Instruktionen in der Programmreihenfolge in die Liste eingefügt werden. (4 Punkte)

Zeit	Liste	P1	P2	P3
1	*1,2,4,5,8*	*1*	*2*	*4*
2	*5,8,3*	*5*	*8*	*3*
3	*6,9*	*6*	*9*	-
4	*7,10*	*7*	*10*	-
5	*11*	*11*	-	-
6	*12*	*12*	-	-
7	*13*	*13*	-	-
8				
9				
10				

i. Ermitteln Sie nun den Plan unter Verwendung der Highest-Level-First-Variante. Bei gleichen Leveln werden die Instruktionen in der Programmreihenfolge verteilt. (5 Punkte)

Zeit	Liste	P1	P2	P3
1	*8,1,2,4,5*	*8*	*1*	*2*
2	*4,5,9,3*	*4*	*5*	*9*
3	*3,6,10*	*3*	*6*	*10*
4	*7,11*	*7*	*11*	-
5	*12*	*12*	-	-
6	*13*	*13*	-	-
7				
8				
9				
10				

b) Scheduling (4+10+4 = 18 Punkte)

i. Erläutern Sie die „Highest Response Ratio Next“ Strategie. (4 Punkte)

Beim Scheduling wird für jeden Prozess ein „Response Ratio“

$$r := \frac{\text{Wartezeit+Bedienzeit}}{\text{Bedienzeit}}$$

berechnet. Der Prozess mit dem größten r rechnet als nächster. Wartende Prozesse sammeln bei diesem Verfahren Punkte in Form ihrer Wartezeit. Das Verfahren bevorzugt Prozesse mit kleinerer Bedienzeit, da deren Response Ration schneller wächst. Es gibt keine Verdrängung.

ii. Vier Prozesse mit ihren Startzeitpunkten und der benötigten Rechenzeit seien gegeben:

Prozess	Startzeitpunkt	Benötigte Rechenzeit
P1	0	3
P2	2	4
P3	4	2
P4	5	1

Nehmen Sie an, dass neu ankommende Prozesse zu ihrem Startzeitpunkt bereits in der „Ready“-Schlange eingegliedert sind. Geben sie die Reihenfolge der Prozesse an, indem Sie die Prozessnummern in die entsprechenden Kästchen eintragen. (8 Punkte)

- nach dem LCFS-PR-Verfahren

0	1	2	3	4	5	6	7	8	9	10	11	12
P1	*P1*	*P2*	*P2*	*P3*	*P4*	*P1*	*P2*	*P2*	*P3*			

- nach dem RR-Verfahren mit einer Zeitscheibengröße von 2. Terminiert der laufende Prozess innerhalb einer Zeitscheibe, so erhält der nächste Prozess eine volle neue Zeitscheibe.

0	1	2	3	4	5	6	7	8	9	10	11	12
P1	*P1*	*P2*	*P2*	*P1*	*P3*	*P3*	*P2*	*P2*	*P4*			

i. Berechnen Sie die Dehnfaktoren der vier Prozesse für eines der beiden Verfahren aus ii. (4 Punkte)

$$\text{Dehnfaktor} = \frac{\text{Verweilzeit}}{\text{Rechenzeit}}$$

LCFS-PR:

$\text{Dehnfaktor}_1 = 7/3$

$\text{Dehnfaktor}_2 = 7/4$

$\text{Dehnfaktor}_3 = 6/2 = 3$

$\text{Dehnfaktor}_4 = 1/1 = 1$

RR:

$\text{Dehnfaktor}_1 = 5/3$

$\text{Dehnfaktor}_2 = 7/4$

$\text{Dehnfaktor}_3 = 3/2$

$\text{Dehnfaktor}_4 = 5/1 = 5$